WSRL-GD-02/91

AR-006-485

TURBO PASCAL/GEM SOFTWARE INTERFACE FOR SCIENTIFIC GRAPH PREPARATION

R.M. THAMM, D.A. GREEN and O.M. WILLIAMS

GUIDED WEAPONS DIVISION
WEAPONS SYSTEMS RESEARCH LABORATORY

Approved for Public Release.

APRIL 1991

DSTO WSRL DEPARTMENT OF DEFENCE
DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

GENERAL DOCUMENT
WSRL-GD-02/91

A-1

TURBO PASCAL/GEM SOFTWARE INTERFACE FOR SCIENTIFIC GRAPH PREPARATION

R.M. Thamm, D.A. Green and O.M. Williams

A B S T R A C T (U)

This document represents a user manual for the Turbo Pascal/GEM software interface that has been developed recently within Guided Weapons Division. The interface has been developed primarily to enable Turbo Pascal programmer to replicate their softcopy graphical output in high quality hardcopy form. The output from the interface is a GEM file that can be edited as required within either GEM Draw or GEM Artline, prior to generation of a PostScript file. A number of mathematical procedures have been included in order to extend the capabilities of Turbo Pascal, together with a useful library of curve fitting procedures. The interface also incorporates a form filling technique for versatile data entry at any point within a Turbo Pascal program. A detailed description of all procedures and functions is included together with the code and graphical output from a number of test programs designed to illustrate the capabilities of the interface.

Author's address:
Guided Weapons Division
Weapons Systems Research Laboratory
PO Box 1700, Salisbury
South Australia

Requests to: Chief, Guided Weapons Division

Table of Contents

PART I – TURBO PASCAL/GEM INTERFACE

PART II – SOFTWARE STRUCTURE

LIST OF APPENDICES

LIST OF TABLES

LIST OF FIGURES

PART I – TURBO PASCAL/GEM INTERFACE

1. INTRODUCTION

With the proliferation of personal computers, scientists and technologists now have unprecedented access to desktop computing power at a level that just a few years ago would have been available only from central mainframe computers. Most notable is the ready availability of graphics procedures within the high level languages Turbo Pascal and Turbo C that considerably enhance the visualisation of scientific output by enabling convenient presentation in graphical form. There has been, however, some lag in the development of software suitable for quality hardcopy production. Although graph generation software is available within various commercial packages such as the GEM software library, it tends to be directed more towards the business user whose interest lies more in bar and pie charts rather than towards scientists and engineers who tend to have a considerably greater interest in line graphs.

In this document, we describe a software package that has been developed within Guided Weapons Division for interfacing Turbo Pascal to the GEM software library. Availability of the package allows the user to develop and present graphical data in standard softcopy form on an IBM PC compatible using Turbo Pascal, and at the same time to produce an output file written in GEM format. This file acts as the input to either GEM Draw or GEM Artline, thus giving the user the capability for editing Turbo Pascal generated graphical data in such a way as to produce hardcopy output in publication quality form, incorporating lettering as required in a wide variety of fonts as well as whatever artistic touches that the author may desire. Even colour hardcopy reproduction is possible through use of an appropriate colour printer. Experience has proven that the TP/GEM interface described herein provides an effective link between scientific programming/graphical data analysis and production of publication quality graphical output.

The TP/GEM interface and the essential requirements enabling its utilization are described in Part I of this document together with a general description of its structure. Part II is written in the form of a user manual and includes a detailed description of all units and procedures. A number of test programs designed specifically for demonstrating the capabilities of the interface are included in the Appendices. A TP/GEM floppy disc is available on request to all interested users within DSTO. For best appreciation of the capabilities of the interface, it is recommended that potential users request the disc.

2. HARDWARE/SOFTWARE REQUIREMENTS

2.1 General

The interface described in this document has been developed primarily to enable Turbo Pascal programmers to replicate their softcopy graphical output in high quality hardcopy form. The available procedures, however, cover rather more than the primary graphical output role. A number of mathematical procedures are included in order to extend the capabilities of Turbo Pascal, together with a useful library of curve fitting procedures. The interface also incorporates a form filling technique for versatile data entry at any point within a program.

It is expected that users will be proficient with Turbo Pascal and will be familiar with the graphical design capabilities of either GEM Draw or GEM Artline. An IBM PC compatible

is required. It should have a full complement of memory (640 KBytes) and have an EGA or VGA Graphics capability. The raw output is in the form of a GEM format file. It is therefore necessary that sufficient disk storage space be available to store the output files. The user must have Turbo Pascal 5 or 5.5 installed in order to use the graphics procedures. In order to later manipulate the graphics produced in GEM format the user must have GEM Artline or GEM Draw installed and a mouse driver. It is expected that the user has PC or MS-DOS V2.1 or a later version installed.

2.2 Memory Configuration

Some of the units described in this document use large amounts of main memory. On occasions, Turbo Pascal will produce the message *'Out of Memory'* and not complete compilation. A number of techniques can be used to resolve this problem :

1. Check that you are compiling to disk rather than memory by pressing Alt-C from the Turbo Pascal Integrated Development Environment (IDE). The pop-up window has a *'Destination'* option. If this shows *'Memory'*, then change it to disk.

2. Reduce the memory requirement. This may involve removing all units that are not required from the *Uses* clause in your program, or by reducing array sizes.

3. The Turbo Pascal editor can be loaded into extended memory (if installed), saving about 64 KBytes of memory. Consult your Turbo Pascal manual for details on how this may be achieved.

4. Try removing memory-resident programs from memory (*e.g.*, Sidekick or other Terminate and Stay Resident (TSR) programs).

5. Exit from the IDE to DOS. Try compiling using the command line compiler. For example, try:

   >TPC /L GRAFTEST.PAS [ENTER]

   where /L indicates 'Link Buffer on Disk', not memory. For other TPC options, type:

   >TPC [ENTER]

   All the command line options will appear on the screen. Alternatively, consult the Turbo Pascal manual.

3. HARD DISK INSTALLATION

1. If you haven't already done so, make a backup of the original disk supplied with this package. The easiest way to do this is to use the DOS diskcopy command (*i.e.*, DISKCOPY A: A: [ENTER]) and follow the prompts.

2. Exit to DOS and load the supplied disk into the floppy disk drive A:

   (a) Change directory to the level above where you want the files to be copied into. For example, in order to place the files in a directory immediateiy below the ROOT directory,

type CD\ [ENTER]. Alternatively, if you prefer to place them in a subdirectory of your Turbo Pascal directory then type CD\TP [ENTER].

(b) Make a directory into which the supplied files can be copied.
i.e., type: MD PLOT [ENTER].

(c) Now move into your new directory.
i.e., type: CD PLOT [ENTER].

3. Copy the files into the new directory.
i.e., type: COPY A:*.* [ENTER].

4. Copy the Turbo Pascal Graphics and Font files into the new directory. These could be anywhere on your disk depending on how your Turbo Pascal was installed. For Turbo 5.5 they are normally located in the \TP directory. In this case type:

```
COPY \TP\*.BGI [ENTER]
COPY \TP\*.CHR [ENTER]
```

Note that you may not want to make a copy of these files. If so, you can alternatively load file PLOTLIB.PAS into your Turbo Pascal editor and locate Procedure *ScreenStart* (about line 158). In this procedure there is a call to *InitGraph* in the Graphics library. Enter the name of the directory in which the .BGI and .CHR files are located between the null quotes and save the file.

e.g.,
```
InitGraph(GraphDriver, GraphMode, '');
```
becomes
```
InitGraph(GraphDriver, GraphMode, 'C:\TP\GRAPHICS');
```

Refer to the Turbo Pascal reference manual for details.

5. It is expected that users have already configured Turbo Pascal within their existing systems. A number of test programs (reproduced in the Appendices) are supplied in order to illustrate the capabilities of the TP/GEM interface. It is suggested that one of these files be loaded into the Turbo Pascal editor and compiled. Generally it is necessary to perform a BUILD when using a unit for the first time; *i.e.*, *ALT-C*, then *B* from the Integrated Development Environment. Alternatively, each unit can be picked up with the editor and compiled separately.

6. **Read Section 4.1 before continuing. It is important that the usage of the type *Float*, which is an integral part of the TP/GEM software structure, is well understood before the package can be successfully applied.**

4. INTERFACE STRUCTURE

The interface software is supplied on a master diskette in the form of a suite of units, each of which is comprised of a number of procedures that together define the functional use of the unit. The units available to be called at the start of the Turbo Pascal program, all of which have a .PAS extension, include:

Unit	*Role*
AXES	Data types and procedures for drawing axes
BEZIER	Bezier curve generation
COORDS	Transformations between coordinate systems
DEFAULTS	Sample Default unit
DSM	Discrete smoothing cubic spline
FLOATDEF	Definition of the type *Float*
FORM	Construction of a flexible form filling system
GEMMETA	Not intended for general use
INTERP	Curve data interpolation and derivatives
MATHLIB	Supplementary mathematical library
PLOTLIB	Line graph drawing procedures
SBEZIER	Smoothed Bezier curve generation
SMOOTH	Noisy data smoothing
TYPEDEF	Global data types

The above list provides a guide as to the versatility of the interface software. Only the major units and their general structures are described here. Part II provides a detailed description of the variables and procedures available to the user. A number of examples of application of the procedures for producing graphical output appear in the Appendices.

4.1 Type *Float*

The TP/GEM interface employs the type *Float* which must be used for all floating point data. This is particularly important when variables are passed into a TP/GEM function or procedure, all of which use the type *Float*. *Float* is defined in the unit FLOATDEF, the default value being *Single*. This value may be modified by the user to either *Double* or *Extended*. Examples of application of *Float* appear throughout the Appendices.

4.2 Arrays

Pascal does not support variable length arrays. However, Turbo Pascal supports untyped parameters that can be used to emulate variable length arrays. When passing arrays to procedures that apply untyped parameters in this way, the following rules must be obeyed:

(a) Always pass the address of the first element of the data you wish to pass.

(b) The array must have elements of the correct type (normally *Float*).

(c) The length parameter must be correct and less than the specified limit.

For example, in order to pass the data contained in the subarray X[20]..X[40], the user must specify X[20] as the array parameter and 21 as the length parameter.

The limit on the number of data elements passed in arrays is set by the constant *MaxPt* (default = 1200) in the unit TYPEDEF. This can be increased if necessary without increasing the memory requirements of the package.

The following units set their own limits on the length of arrays passed to them:

INTERP This unit defines the constant *TNArraySize* (default = 201). Since the unit also declares variables of the type ARRAY[0..*TNArraySize*] OF *Float*, altering the value of *TNArraySize* will change the memory requirements.

SBEZIER This unit defines the constant *InPts* (default = 500). Since the unit also declares variables of the type ARRAY[1..*InPts*] OF *Float*, altering the value of *InPts* will change the memory requirements of the unit.

SMOOTH The unit SMOOTH defines a constant *MaxPt* (default = 8192) which must be an integral power of two, and greater than or equal to $(n + 2 * pts)$ where n and pts are parameters of the procedure *SmoothFt*. If *TYPEDEF.MaxPt* is increased, the user is advised to check that *SMOOTH.MaxPt* is still large enough.

4.3 Mathematical library

The common mathematical functions such as Power, Tan, ArcSin, ArcCos and Log10 that are not provided within Turbo Pascal are included within the unit MATHLIB. Reciprocal conversion between degree and radian measure is also included, as are functions for determining the mantissa and exponent of a floating point number and a procedure for locating array minima and maxima.

4.4 Form filling library

Unit FORM includes procedures that assist in the versatile entry of constant data into a program. It is a simple matter to set up a form that is displayed on the monitor and which contains details of all the parameters (String, Integer or Floating Point) that the user may wish to set before the program runs. At this stage the user is able to change at will the default values that are automatically displayed, without the need to modify the existing code. While not mandatory, usage of unit FORM represents a convenience that becomes quickly apparent when a program needs to be run many times with different starting parameters.

4.5 Plot library

The units that comprise the plot library — namely, AXES, COORDS and PLOTLIB — represent the heart of the software interface. It is within these units that the procedures exist for creating the graphical output in the form required by the user, available both on screen and as a GEM format output file.

4.5.1 Coordinate systems

The Plot Library employs three different coordinate systems:

World Coordinate Space

This is the user-defined coordinate system. The user specifies a rectangular window within this space by applying the *DefineWorld* procedure. Each user-defined window is called a World and is defined by the lower left (ll) and upper right (ur) corners. Only data inside the window will be plotted. The mapping of a World into the plot box is illustrated in Figure 1. The user can define many Worlds, thus allowing several sets of data to be incorporated within the one plot, as illustrated in Appendix II.

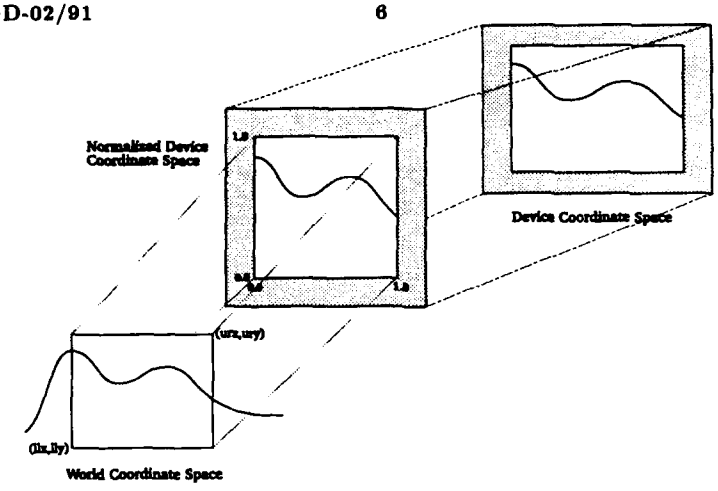


Figure 1 TP/GEM coordinate systems

Normalized Device Coordinate (NDC) Space

This is a standard coordinate system that is used for all internal calculations. Each World is mapped into the unit square as shown in Figure 1. All curves are clipped into the unit square. All numbering, labelling and ticking is performed in NDC space.

Device Coordinate Space

Each output device is characterized by its own coordinate space. The TP/GEM package supports two devices: the screen and the GEM file. The screen coordinate space varies with the screen type (*e.g.*, EGA or VGA). The output plot resides in a rectangular area. The plot box is centred in this area, the relative size of the box being set by the *ScaleFactor* parameter within the *ScreenStart* and *GEMStart* procedures. For the screen case, the rectangular area corresponds to the entire screen. For the GEM metafile case, the area will fit onto an A4 page, the dimensions being set in the ratio 4:3 for the *Landscape* and *Portrait* options and 1:1 for the *SquarePage* option.

Of prime importance to users is the definition and use of Worlds. A World is defined using the *DefineWorld* procedure. The user specifies the lower left and upper right corners of this World. *DefineWorld* returns an integer value which represents the World. This value should be passed to all the axes drawing, labelling and plotting procedures, as illustrated within the appendices. Most of the procedures in this package require data in the user-defined World coordinates.

A few procedures require data in NDC coordinates (*e.g.*, *PlotDashedLine*). The user has therefore been given access to procedures that allow conversion between the different coordinate systems. An example of the use of NDC coordinates is included in Appendix II. Some constants (*e.g.*, *TickLen*) are defined in pseudo-NDC space. The default tick length of 0.02 means that the tick marks will be sized at 0.02 times the longer side of the plot box.

4.6 Curve fitting library

As a supplement to the plot library, a curve fitting library comprised of the units BEZIER, SBEZIER, DSM, INTERP and SMOOTH is included for the convenience of the user. As the

names suggest, the primary curve data may be manipulated before the output curve is plotted. The data may, for example, be interpolated or smoothed before plotting. Only a few data points may be available, in which case the procedures within BEZIER or SBEZIER may be applied in order to generate smooth connecting curves. The availability of these procedures adds significantly to the versatility and general capability of the TP/GEM interface.

4.6.1 Bezier Curves

GEM Artline supports compound Bezier Curves[1]. Each segment of a compound Bezier Curve is defined by four points: two endpoints (x_0, y_0) and (x_3, y_3) and two control points (x_1, y_1) and (x_2, y_2). The parametric equations of a Bezier segment are:

$$x(t) = x_0(1-t)^3 + x_1 t(1-t)^2 + x_2 t^2(1-t) + x_3 t^3 \quad (1)$$

and

$$y(t) = y_0(1-t)^3 + y_1 t(1-t)^2 + y_2 t^2(1-t) + y_3 t^3. \quad (2)$$

Two units, BEZIER and SBEZIER, have been provided for converting user data into composite Bezier curves. The advantages of Bezier curves over normal 'curves' composed of straight line segments are:

i) Beziers are real curves that are not composed of straight line segments.

ii) Compound Bezier Curves are single entities in GEM Artline. Normal segmented curves are limited to 100 line segments; curves of more than 100 segments appear to GEM Artline as several objects grouped together.

iii) Bezier Curves can be plotted in dashed line form. However, the single entity nature in GEM Artline is then lost.

iv) Bezier curves can be edited with ease in GEM Artline.

Two separate units have been provided for plotting Bezier curves:

(1) The unit BEZIER generates quick curves and requires less memory than the alternative unit, SBEZIER. BEZIER selects up to 40 of the user-input data points as the endpoints of the Bezier segments. Data points that are not chosen as endpoints may not lie on the resulting curve. Furthermore, in some cases the resulting curve may not fit the input data very well, particularly when large curvatures are involved.

(2) The unit SBEZIER is more sophisticated. This unit generates a smoothing spline for the input data and selects the endpoints for the Bezier segments from the spline. The Bezier segment equations are found by forcing the first and second derivatives of the Bezier curves to match those of the smoothing spline at the segment endpoints. The unit produces better results than BEZIER but at the expense of greater memory preferences and slower running time. Dashed lines of higher quality (dashes of equal length) are generated and several dashed line styles are supported.

PART II - SOFTWARE STRUCTURE

In this second part, the structure of the TP/GEM interface is detailed. Sufficient information is given to allow a progammer to interface between Turbo Pascal and GEM and to thereby produce laser printer output of the graphical data displayed initially on the PC screen.

Examples of programs that usefully illustrate many features of the interface software are reproduced within the Appendices and are also included on the diskette. Note that some minor displacements in the positioning of the labels appearing in the raw GEM file output have been corrected within each of the graphs shown in the Appendices. The GEM colours (which do not exactly match the Turbo Pascal softcopy colours) have also been set universally to black (except in the case of the output from SMOOTHTEST.PAS).

5. UNITS

The interface structure is divided into a number of units, each of which has a .PAS extension. It is advisable to call the units FLOATDEF and TYPEDEF in all programs. The available units are:

Unit	*Section*	*Role*
AXES	6.3	Data types and procedures for drawing axes
BEZIER	6.5	Bezier curve generation
COORDS	6.8	Transformations between coordinate systems
DEFAULTS	6.12	Sample default unit
DSM	6.7	Discrete smoothing cubic spline
FLOATDEF	6.10	Definition of the type *Float*
FORM	6.4	Construction of a flexible form filling system
GEMMETA	—	Not intended for general use
INTERP	6.9	Curve data interpolation and derivatives
MATHLIB	6.1	Supplementary mathematical library
PLOTLIB	6.2	Line graph drawing routines
SBEZIER	6.5	Smoothed Bezier curve generation
SMOOTH	6.6	Noisy data smoothing
TYPEDEF	6.11	Global data types

The complete list of available procedures and functions within the units is given in Table 1 (in alphabetical order).

5.1 Data types and global variables

The units incorporate a number of global variables, the default values of which can be modified at any point within the user source code, as illustrated in Appendices I to III. Alternatively, the required changes to the default values can be entered into the unit DEFAULTS.PAS, thus enabling the TP/GEM interface to be tailored to the user requirements. Note that the units in which the appropriate global variables are defined must then be called under the *USES* heading within DEFAULTS.PAS. The global variables and data types are listed in Tables 2 and 3 respectively.

Table 1 AVAILABLE PROCEDURES AND FUNCTIONS

Procedure or Function	*Unit*	*Procedure or Function*	*Unit*
AdvanceLine	Form	LogRightAxis	Axes
ArcCos	MathLib	LogScale	Axes
ArcSin	MathLib	LogXAxis	Axes
AxisScale	Axes	LogYAxis	Axes
ClearCoords	Coords	Mantissa	MathLib
CloseForm	Form	NameRightAxis	Axes
CubicSplineFree	Interp	NameTopAxis	Axes
DefineWorld	Coords	NameXAxis	Axes
Deg	MathLib	NameYAxis	Axes
Derivative	Interp	NDC	Coords
DevCoords	Coords	NDCX	Coords
DevX	Coords	NDCY	Coords
DevY	Coords	OpenForm	Form
DrawAxes	Axes	PlotBezier	Bezier
Exponent	MathLib	PlotCurve	PlotLib
FindExtrema	MathLib	PlotDashedLine	PlotLib
FitCS	DSM	PlotEnd	PlotLib
FitDS	DSM	PlotMarkers	PlotLib
FitDDS	DSM	Power	MathLib
FitDSm	DSM	Rad	MathLib
FloatItem	Form	ScreenStart	PlotLib
GemWidth	PlotLib	SecondDerivative	Interp
GemStart	PlotLib	SetDefaults	Defaults
GemTextSettings	PlotLib	SmoothBezier	SBezier
GetIntegerValue	Form	SmoothFt	Smooth
GetRealValue	Form	StringItem	Form
GetStringValue	Form	Tan	MathLib
IntegerItem	Form	VarLabelRight	Axes
LabelRightAxis	Axes	VarLabelTop	Axes
LabelXAxis	Axes	WorldX	Coords
LabelYAxis	Axes	WorldY	Coords
Log10	MathLib		

Table 2 GLOBAL VARIABLES

Variable name		*Type*	*Default*	*Unit*
AxisBox	=	AxisOption;	FullBox	Axes
AxisColor	=	WORD;	Green	Axes
AxisTicks	=	TickOption;	TwoSidesTicked	Axes
DeviceList	=	OutputDevicesList;	—	TypeDef
FormControl	=	INTEGER;	1	Form
LabelMargin	=	Float;	0.02	Axes
LogTicks	=	LogTickOption;	UnitsTicked	Axes
Side	=	SideType;	OutSide	Axes
TickLen	=	Float;	0.02	Axes

Table 3 DATA TYPES

Data Type			*Unit*	Section
AxisOption	=	(FullBox, HalfBox);	Axes	6.3
Float	=	REAL;	FloatDef	6.10
GemFontType	=	(Swiss, SwissBold, Dutch, DutchBold, Charter)	PlotLib	6.2
GraphType	=	(Linear, Log, LogLinear, LinearLog);	TypeDef	6.11
LineType	=	(Dashed, ShortDashed);	PlotLib	6.2
LogTickOption	=	(DecadesTicked, UnitsTicked);	Axes	6.3
MarkerType	=	(Dot, Plus, Star, Square, Cross, Diamond);	PlotLib	6.2
OrientType	=	(Portrait, LandScape, SquarePage);	PlotLib	6.2
OutputDevicesList	=	Set of OutputDisplayType;	TypeDef	6.11
OutputDisplayType	=	(Screen, GemFile);	TypeDef	6.11
PlotArray	=	Array[1..MaxPt] of Float;	TypeDef	6.11
SideType	=	(Inside, Outside);	Axes	6.11
TickOption	=	(OneSideTicked, TwoSidesTicked);	Axes	6.11
TNVector	=	Array[0..TNArraySize] of Float;	DSM	6.7
WidthType	=	(FullWidth, HalfWidth);	PlotLib	6.2

2. UNITS AND PROCEDURES

2.1 Unit MathLib

The MathLib unit contains standard mathematical procedures that are not supplied within Turbo Pascal.

Power

Header : FUNCTION Power(a, b : Float) : Float;
Purpose : Returns the value of a to the power of b.
Example : c := Power(2, 8);
c will assume the value of 256 or 2^8.

Tan

Header : FUNCTION Tan(x: Float) : Float;
Purpose : Returns the tangent of x where x is in radians.
Example : c := Tan($\pi/4$);
c will assume the value of 1.0.

Arcsin

Header : FUNCTION ArcSin(x: Float) : Float;
Purpose : Returns the arcsine of x.
Example : c := ArcSin($1/\sqrt{2}$);
c will assume the value of $\pi/4$ radians.

ArcCos

Header : FUNCTION ArcCos(x: Float) : Float;
Purpose : Returns the arccosine of x.
Example : c := ArcCos($\sqrt{3}/2$);
c will assume the value of $\pi/6$ radians.

Rad

Header : FUNCTION Rad(x: Float) : Float;
Purpose : Converts degrees to radians.
Example : c := Rad(45.0);
c will assume the value of $\pi/4$ radians.

Deg

Header : FUNCTION Deg(x: Float) : Float;
Purpose : Converts radians to degrees.
Example : c := Deg(3.1416);
c will assume the value of 180°.

Log10

Header : FUNCTION Log10(x: Float) : Float;
Purpose : Returns $\log_{10}(x)$.
Example : c := Log10(1000);
c will assume the value of 3.

Exponent

Header : FUNCTION Exponent(x: Float) : Float;
Purpose : Returns the exponent of a floating point number.
Example : c := Exponent(1234.567);
c will assume the value of 3 (*i.e.*, $1234.567 = 1.234567 \times 10^3$).

Mantissa

Header : FUNCTION Mantissa(x: Float) : Float;
Purpose : Returns the mantissa of a real number.
Example : c := Mantissa(1234.567);
c will assume the value of 1.234567 (*i.e.*, $1234.567 = 1.234567 \times 10^3$).

FindExtrema

Header : PROCEDURE **FindExtrema**(VAR x; Npts : INTEGER;
VAR MinValue, MaxValue : Float; VAR imin, imax : INTEGER);
Purpose : Finds the minimum and maximum values of an array.
Also returns the index positions where the minima and maxima were found.
Example : FindExtrema(PlotData, 100, Minimum, Maximum, MinPos, MaxPos);
Returns the maximum and minimum values of the first 100 elements
of the array *PlotData*. The minimum and maximum values are
located at index positions *MinPos* and *MaxPos* respectively.

6.2 Unit PlotLib

Unit PlotLib contains line graph drawing procedures to screen and/or to a GEM metafile.

6.2.1 Enumerated data types

```
OrientType    =  (Portrait, LandScape, SquarePage);
GemFontType   =  (Swiss, SwissBold, Dutch, DutchBold, Charter);
LineType      =  (Dashed, ShortDashed);
WidthType     =  (FullWidth, HalfWidth);
MarkerType    =  (Dot, Plus, Star, Square, Cross, Diamond);
```

6.2.2 Mandatory Calls

The following list includes the mandatory calls for drawing a curve:

ScreenStart/GemStart	:	Initialise screen or GEM file
DrawAxes	:	Draws rectangle around graph area
DefineWorld	:	Define World (user) coordinates
LabelXAxis or LogXAxis(unit Axes)	:	Draw tick marks and number X Axis
LabelYAxis or LogYAxis(unit Axes)	:	Draw tick marks and number Y Axis
PlotCurve (or PlotBezier)	:	Draw a curve
PlotEnd	:	Finish plot

6.2.3 Available procedures and functions

GemWidth	:	Sets line width for Gem polylines
GemStart	:	Initialises Gem plot file (GEMfile)
GemTextSettings	:	Sets font size and type for Gem file text
PlotCurve	:	Plots the contents of an array
PlotDashedLine	:	Sets the color, dash type and line width
PlotEnd	:	Called when plot is complete
PlotMarkers	:	Used to draw data points
ScreenStart	:	Used to initialise and scale plotting to screen

6.2.4 Procedure and function headers

Reference should be made to the above enumerated data types.

PROCEDURE **GemWidth**(LineWidth : INTEGER);

LineWidth	:	Width of GEM line (default value = 50 (0.5 mm).

PROCEDURE **GemStart**(FileName : STRING; Orientation : OrientType;
ScaleFactor : Float);

FileName	:	Name of GEM output file. Call this file from GEM draw
	:	or GEM Artline to access your graphics output.
Orientation	:	Orientation of output as per type *OrientType*.
ScaleFactor	:	Defines the fraction of the available plot dimension within the plot rectangle.

PROCEDURE **GemTextSettings** (Font : GemFontType; FontSize : INTEGER);

Font	:	Specifies font type as per type*GemFontType* (default = Swiss).
FontSize	:	Point size for font (default = 17 pt).

PROCEDURE **PlotCurve**(Color : WORD; NPoints : INTEGER; GraphKind : GraphType; VAR x, y; World : INTEGER);

Color	:	Color of plotted curve (standard Turbo Pascal screen colors).
NPoints	:	Number of points in curve.
GraphKind	:	(Linear, Linearlog, LogLinear, Log).
x, y	:	Arrays containing the curve (x, y) coordinates.
World	:	The World associated with the curve.

PROCEDURE **PlotDashedLine**(Color : WORD; LineForm : LineType; DashedWidth : WidthType; x1, y1, x2, y2 : Float);

Color	:	Color of plotted curve.
LineForm	:	Dashed or ShortDashed.
DashedWidth	:	FullWidth or HalfWidth.
x1, y1	:	One end point of the line (NDC Coordinates).
x2, y2	:	Other end point of the line (NDC Coordinates).

PROCEDURE **PlotEnd**;

PROCEDURE **PlotMarkers**(Color : WORD; Kind : MarkerType; Size : Float; NPts : INTEGER; GraphKind : GraphType; Var x, y; World : INTEGER);

Color	:	Color of plotted curve.
Kind	:	Type of marker (see Marker type above).
Size	:	Size of marker relative to standard (=1) device size.
Npts	:	Number of markers (or points) to plot.
GraphKind	:	(Linear, Linearlog, LogLinear, Log).
x, y	:	Arrays containing the x and y values of the points to plot.
World	:	The World associated with the curve.

PROCEDURE **ScreenStart**(ScaleFactor : Float);

ScaleFactor	:	Fraction of full screen dimensions allocated for the plot rectangle.

6.3 Unit Axes

Unit Axes contains data types and procedures for drawing axes on graphs. The unit also contains an autoscaling procedure that may be applied separately to all curve axes.

6.3.1 Available structures

6.3.1.1 Data types

The following descriptive datatypes are used to describe the axes required:

```
TickOption      =   (OneSideTicked, TwoSidesTicked);
SideType        =   (Inside, Outside);
LogTickOption   =   (DecadesTicked, UnitsTicked);
AxisOption      =   (FullBox, HalfBox);
Array100        =   ARRAY[1..100] OF Float;
```

6.3.1.2 Global variables and defaults

AxisBox	:	AxisOption;	*FullBox*
AxisColor	:	WORD;	*Green*
AxisTicks	:	TickOption;	*OneSideTicked*
LabelMargin	:	Float;	*0.02*
LogTicks	:	LogTickOption;	*UnitsTicked*
Side	:	SideType;	*OutSide*
TickLen	:	Float;	*0.02*

Note that a grid can be established by setting *Side* to *InSide* and *TickLen* to 1.0. The width of the grid lines may be set through use of the procedure *GEMWidth*, as illustrated within Appendix III.

6.3.2 Available procedures

AxisScale	:	Provides autoscaled values from specified min and max values.
LabelRightAxis	:	Draws ticks and numbers on the right axis.
LabelXAxis	:	Draws ticks and numbers on the bottom axis and optionally draws ticks on the top axis.
LabelYAxis	:	Draws ticks and numbers on the left axis and optionally draws ticks on the right axis.
LogRightAxis	:	As for LabelRightAxis except with log scale.
LogScale	:	Provides autoscaled logarithmic values from specified maximum value and required number of decades.
LogXAxis	:	As for LabelXAxis except with log scale.
LogYAxis	:	As for LabelYAxis except with log scale.
NameRightAxis	:	Writes a name on the right axis.
NameTopAxis	:	Writes a name on the top axis.
NameXAxis	:	Writes a name on the bottom axis.
NameYAxis	:	Writes a name on the left axis.
VarLabelRight	:	Draws user-defined ticks and labels on the right axis.
VarLabelTop	:	Draws user-defined ticks and labels on the top axis.

6.3.3 Procedure headers

PROCEDURE **AxisScale**(AxisnameStr : STRING; VAR steps : INTEGER;
VAR min, max : Float; VAR Decimals : INTEGER);

AxisnameStr	:	Identifying name for axis.
Steps	:	Number of steps required on the axis.
min, max	:	Minimum and maximum values (input as specified, output as autoscaled or user-selected).
Decimals	:	Number of displayed decimals.

PROCEDURE **LabelRightAxis**(LabelColor : Word; YIntervals, decimals : INTEGER; World : INTEGER);

LabelColor	:	Color assigned to the ticks and numbers.
YIntervals	:	Total intervals between right hand axis ticks.
Decimals	:	Number of displayed decimals.
World	:	The world associated with the plotted curve.

PROCEDURE **LabelXAxis**(LabelColor : WORD; XIntervals, Decimals : INTEGER; World : INTEGER);

LabelColor	:	Color assigned to the ticks and numbers.
XIntervals	:	Total intervals between bottom axis ticks.
Decimals	:	Number of displayed decimals.
World	:	The world associated with the plotted curve.

PROCEDURE **LabelYAxis**(LabelColor : WORD; YIntervals, Decimals : INTEGER; World : INTEGER);

LabelColor	:	Color assigned to the ticks and numbers.
YIntervals	:	Total intervals between left hand axis ticks.
Decimals	:	Number of displayed decimals.
World	:	The world associated with the plotted curve.

PROCEDURE **LogRightAxis**(LabelColor : WORD; World : INTEGER);

LabelColor	:	Color assigned to the right hand ticks and numbers.
World	:	The world associated with the plotted curve.

PROCEDURE **LogScale**(AxisnameStr : STRING; VAR min, max : Float; NoDecades: INTEGER);

AxisnameStr	:	Identifying name for axis.
min, max	:	Minimum and maximum values — max is input as specified, output as autoscaled or user-selected. — min is calculated from *NoDecades* and *max*
NoDecades	:	Number of decades required on log axis.

PROCEDURE **LogXAxis**(LabelColor : WORD; World : INTEGER);

LabelColor	:	Color assigned to the bottom axis ticks and numbers.
World	:	The world associated with the plotted curve.

PROCEDURE **LogYAxis**(LabelColor : WORD; World : INTEGER);

LabelColor	:	Color assigned to the left hand axis ticks and numbers.
World	:	The world associated with the plotted curve.

PROCEDURE **NameRightAxis**(Color : WORD; YNameStr : STRING);

Color	:	Color assigned to the right axis name.
YNameStr	:	Right axis name.

PROCEDURE **NameTopAxis**(Color : WORD; XNameStr : STRING);

Color	:	Color assigned to the top axis name.
XNameStr	:	Top axis name.

PROCEDURE **NameXAxis**(Color : WORD; XNameStr : STRING);

Color	:	Color assigned to the bottom axis name.
XNameStr	:	Bottom axis name.

PROCEDURE **NameYAxis**(Color : WORD; YNameStr : STRING);

Color	:	Color assigned to the left hand axis name.
YNameStr	:	Left hand axis name.

The following two procedures give the opportunity for user-defined labelling. There can be cases where an axis needs to be labelled linearly in terms of a primary parameter on one side, and nonlinearly in terms of a functionally related parameter on the other side, as illustrated in Appendix II. The user must specify two arrays; *MyLabels*, for defining the required labels to be displayed nonlinearly on the axis of interest, and *TickPosn*, which is derived by converting the *MyLabels* values into the corresponding linear parameter values, in accordance with the required nonlinear functional form. For example, if the primary parameter is wavelength λ to be displayed linearly on the lower x-axis as in the example of Appendix II, and the user wishes to display the energy

$$E = \frac{1.240}{\lambda} \tag{1}$$

across the upper x-axis, then the array *MyLabels* defining the required gradations of E is formed, followed by the *TickPosn* array which defines the complementary values of λ obtained from the inverse form of the functional expression (1) above. This example is coded in Appendix II where it may be noted that both *TickPosn* and *MyLabels* are specified as arrays of *Float*.

PROCEDURE **VarLabelRight**(LabelColor : WORD; NTicks : INTEGER;
VAR TickPosn, MyLabels; Decimals : INTEGER; YMin, YMax : Float);

LabelColor	:	Color of right axis label, ticks and numbers.
NTicks	:	Number of ticks to display on right axis.
TickPosn	:	Array of Y coordinates where ticks are to be placed.
MyLabels	:	Array of labels for tick marks.
Decimals	:	Number of displayed decimals.
YMin, YMax	:	Minimum and Maximum values of Y for the right axis.

PROCEDURE **VarLabelTop**(LabelColor : WORD; NTicks : INTEGER;
VAR TickPosn, MyLabels; Decimals : INTEGER; XMin, XMax : Float);

LabelColor	:	Color of right axis label, ticks and numbers.
NTicks	:	Number of ticks to display on right axis.
TickPosn	:	Array of Y coordinates where ticks are to be placed.
MyLabels	:	Array of labels for tick marks.
Decimals	:	Number of displayed decimals.
XMin, XMax	:	Minimum and Maximum values of X for the top axis.

6.4 Unit Form

Unit Form provides procedures for flexible data entry on the same basis as filling out a form. The user constructs the desired layout of the form by application of the available procedures. The input data may be in the form of strings, integers or floating point numbers. Use of the CURSOR or ENTER keys allows movement between different fields and gives the opportunity for the user to change the programmed default data. Data entered in incorrect numeric format is ignored. The data on the form is accepted when either of the PGDN or ESC keys is pressed. Refer to Appendix VII for a detailed description of unit Form within the test program FORMTEST.PAS.

6.4.1 Available procedures

AdvanceLine	:	Inserts a blank line in the form.
CloseForm	:	Indicates that the layout is complete.
FloatItem	:	Places a field for floating point input at the required location.
GetIntegerValue	:	Inputs an integer.
GetRealValue	:	Inputs a floating point number.
GetStringValue	:	Inputs a string.
IntegerItem	:	Places a field for an integer input at the required location.
OpenForm	:	Starts construction of the layout.
StringItem	:	Places a field for a string input at the required location.

6.4.2 Procedure headers

PROCEDURE **AdvanceLine**;

PROCEDURE **CloseForm**;

PROCEDURE **FloatItem**(FloatVal : Float; Decimals : INTEGER; ItemStr : STRING);

FloatVal	:	Float variable name for the field.
Decimals	:	Number of displayed decimals.
ItemStr	:	Prompt associated with the variable name.

PROCEDURE **GetIntegerValue** (VAR IntegerValue : INTEGER);

IntegerValue	:	Returns the integer value input at the keyboard.

PROCEDURE **GetRealValue** (VAR RealValue : Float);

RealValue	:	Returns the real (or Float) value input at the keyboard.

PROCEDURE **GetStringValue** (VAR StringVal : STRING);

StringVal	:	Returns the string input at the keyboard.

PROCEDURE **IntegerItem** (IntegerVal, Width : INTEGER; ItemStr : STRING);

IntegerVal	:	Integer variable to be used with this field.
Width	:	Width of displayed field.
ItemStr	:	Prompt associated with the variable name.

PROCEDURE **OpenForm**(HeadingStr : STRING);

HeadingStr	:	Title for top of form.

PROCEDURE **StringItem** (StringVal : STRING; width : INTEGER; ItemStr : STRING);

StringVal	:	String variable name to be used with this field.
ItemStr	:	Prompt to associate with this variable name.

6.5 Units Bezier and SBezier

These units contain procedures for generating Bezier curves. Bezier curves, which are supported by GEM Artline but not by GEM Draw, are used for drawing smooth curves through a limited number of data points. An example of use of PlotBezier can be found in file TESTDSM.PAS (reproduced within Appendix V). An example of use of SmoothBezier is to be found in file GRAFTEST.PAS (reproduced within Appendix III).

6.5.1 Data type (Available only with SBEZIER)

DashedLineComponent = (LongDash, ShortDash, Dot, LongSpace, ShortSpace);

6.5.2 Available procedures

PlotBezier	:	(Unit Bezier) Plots a Bezier curve.
SmoothBezier	:	(Unit SBezier) Fits to a smoothing Bezier curve. SBezier provides a better fit, the lengths of the dashed segments tend to be more uniform, but it is slower and uses more memory.

6.5.3 Procedure headers

PROCEDURE **PlotBezier** (Color, NPts, NDashes : INTEGER; GraphKind : GraphType; VAR x, y, wx, wy; World : INTEGER);

Color	:	Color of the Bezier curve plot.
NPts	:	Number of input data points.
NDashes	:	Number of dashes if dashed curve is required
	:	(= 0 for a continuous curve).
GraphKind	:	(Linear, Linearlog, LogLinear, Log).
x, y	:	Arrays containing the (x,y) input curve data.
wx, wy	:	Work arrays (must be of dimension NPts).
World	:	The world associated with the Bezier curve.

PROCEDURE **SmoothBezier**(Color : WORD; NPts, LineType : INTEGER; GraphKind : GraphType; VAR x, y; World : INTEGER);

Color	:	Color of the Bezier curve plot.
NPts	:	Number of input data points.
LineType	:	1 — Long dashes and short spaces. 2 — Short dashes and short spaces. 3 — Dots with short spaces. 4 — Medium dashes with short spaces. 5 — Long dash, short space, dot, short space. 6 — Short dashes with long spaces. 7 — Dots with long spaces.
GraphKind	:	(Linear, Linearlog, LogLinear, Log).
x, y	:	Arrays containing the (x,y) input curve data.

6.6 Unit Smooth

This unit is available for smoothing large amounts of noisy curve data. The data values in the x array must be equally spaced. The procedure *SmoothFt*[2] operates by applying a Fast Fourier Transform to the data in the y array, followed by a low pass filter. Linear trends are retained[2].

Program SMOOTHTEST.PAS (Appendix VI) contains an example of application. In this program a sine curve is plotted, random noise is added, and the original curve compared to the smoothed curve obtained by application of *SmoothFt*.

6.6.1 Procedure header

PROCEDURE **SmoothFt**(VAR y; n : INTEGER; Pts : Float);

y	:	Unsmoothed input and smoothed output y array.
n	:	Number of elements in array y (maximum = 8192).
Pts	:	Number of points over which the data is to be smoothed 0 — No Smoothing $> 0.5n$ — Both data and noise are smoothed

6.7 Unit DSM

Unit DSM, which is useful for smoothing moderate amounts of noisy curve data, is based on the algorithm ALG547 — Discrete Smoothing Cubic Spline. Credits and references are included in the Unit source code. An example of use can be found in file TESTDSM.PAS (reproduced within Appendix V).

Within the DSM procedures, parameters *NInterp*, *xs*, *ys*, *b*, *c* and *d* define the spline. The procedures should be called in the following order:

1. FitDSm (To evaluate the spline)
2. FitCS (optional)
3. FitDS (optional)
4. FitDDS (optional)

6.7.1 Available procedures

FitCS	:	Evaluates a cubic spline at each point in an array.
FitDS	:	Evaluates the first derivative of the cubic spline.
FitDDS	:	Evaluates the second derivative of the cubic spline.
FitDSm	:	Computes the discrete natural cubic spline defined over an interval.

6.7.2 Procedure headers

PROCEDURE **FitCS** (NPts, NInterp : INTEGER; VAR x, y, xs, ys, b, c, d);

NPts	:	Number of points to interpolate.
NInterp	:	Number of nodes (x) and data values (y).
x	:	Array of x coordinates for interpolation.
y	:	Returned values of y.
xs	:	Array of floats containing nodes ($x[i] < x[i+1]$).
ys	:	Array of floats containing the smoothed values of data $y[i]$.
b	:	Array of floats containing coefficients for terms $(t - x[i])$ (where t is the interval parameter defined within ALG547).
c	:	Array of floats containing coefficients for terms $(t - x[i])^2$.
d	:	Array of floats containing coefficients for terms $(t - x[i])^3$.

PROCEDURE **FitDS** (NPts, NInterp : INTEGER; VAR x, y, xs, ys, b, c, d);

NPts	:	Number of points to interpolate.
NInterp	:	Number of nodes (x) and data values (y).
x	:	Array of x coordinates for interpolation.
y	:	Returned values of dy/dx.
xs	:	Array of floats containing nodes ($x[i] < x[i+1]$).
ys	:	Array of floats containing the smoothed values of data $y[i]$.
b	:	Array of floats containing coefficients for terms $(t - x[i])$ (where t is the interval parameter defined within ALG547).
c	:	Array of floats containing coefficients for terms $(t - x[i])^2$.
d	:	Array of floats containing coefficients for terms $(t - x[i])^3$.

PROCEDURE **FitDDS** (NPts, NInterp : INTEGER; VAR x, y, xs, ys, b, c, d);

NPts	:	Number of points to interpolate.
NInterp	:	Number of nodes (x) and data values (y).
x	:	Array of x coordinates for interpolation.
y	:	Returned values of d^2y/dx^2.
xs	:	Array of floats containing nodes ($x[i] < x[i+1]$).
ys	:	Array of floats containing the smoothed values of data $y[i]$.
b	:	Array of floats containing coefficients for terms $(t - x[i])$ (where t is the interval parameter defined within ALG547).
c	:	Array of floats containing coefficients for terms $(t - x[i])^2$.
d	:	Array of floats containing coefficients for terms $(t - x[i])^3$.

PROCEDURE **FitDSm** (h, Rho : Float; n : INTEGER; VAR x, y, Wgs, ys, b, c, d);

h	:	Step size used for discrete cubic spline.
Rho	:	Positive parameter for varying smoothness of fit.
	:	— Large Rho emphasizes smoothness
	:	— Small Rho emphasizes fitting
n	:	Number of nodes (x) and data values (y).
x, y	:	Array of floats values containing nodes ($x[i] < x[i+1]$).
Wgs	:	Array of floats containing weights $Wgs[i]$ corresponding to the data $(x[i], y[i])$.
ys	:	Array of floats containing the smoothed values of data $y[i]$.
b	:	Array of floats containing coefficients for terms $(t - x[i])$ (where t is the interval parameter defined within ALG547).
c	:	Array of floats containing coefficients for terms $(t - x[i])^2$.
d	:	Array of floats containing coefficients for terms $(t - x[i])^3$.

6.8 Unit Coords

This unit contains routines for performing transformations between coordinate systems. A maximum of 10 worlds and 10 device spaces can be used. Examples of use of the unit can be found throughout the example files given in the appendices to this document.

6.8.1 Available procedures

ClearCoords	:	Deletes all world and device spaces.
DefineWorld	:	Defines a world coordinate system. World points can be mapped into NDC space.
DevCoords	:	Converts a point in NDC space into device space.
DevX	:	Converts an x coordinate in NDC space into device space.
DevY	:	Converts a y coordinate in NDC space into device space.
NDC	:	Converts a point in world space into NDC space.
NDCX	:	Converts an x coordinate in World space into NDC space
NDCY	:	Converts a y coordinate in World space into NDC space
WorldX	:	Converts an x coordinate in NDC space into world space.
WorldY	:	Converts a y coordinate in NDC space into world space.

6.8.2 Procedure and function headers

PROCEDURE **ClearCoords**;

PROCEDURE DefineWorld(VAR World : INTEGER; llx, lly, urx, ury : Float);

World	:	*Index name for new world.*
llx, lly	:	Lower left coordinate of world coordinate space.
urx, ury	:	Upper right coordinate of world coordinate space.

PROCEDURE **DevCoords**(Device : INTEGER; u, v : Float; VAR x, y : INTEGER);

Device	:	Index number for device.
u,v	:	NDC device coordinates.
x, y	:	NDC coordinate converted into device coordinates.

PROCEDURE **DevX**(Device : INTEGER; u : Float) : INTEGER;

Device	:	Index number for device.
u	:	x coordinate in NDC space.
		— returns x coordinate converted to device space.

PROCEDURE **DevY**(Device : INTEGER; u : Float) : INTEGER;

Device	:	Index number for device.
u	:	y coordinate in NDC space.
		— returns y coordinate converted to device space.

PROCEDURE **NDC**(World : INTEGER; x, y : Float; VAR u, v : Float);

World	:	Index number for world.
x, y	:	Point in world coordinate space.
u, v	:	Point (x,y) converted to NDC coordinates.

FUNCTION **NDCX**(World : INTEGER; x : Float) : Float;

World	:	Index number for world.
x	:	x value in world space.
		— returns x converted to NDC space.

FUNCTION **NDCY**(World : INTEGER; y : Float) : Float;

World	:	Index number for world.
y	:	y value in world space.
		— returns y converted to NDC space.

FUNCTION **WorldX**(World : INTEGER; x : Float) : Float;

World	:	Index number for world.
x	:	x coordinate in NDC space.
		— returns x converted to world space.

6.9 Unit Interp

This unit provides procedures[3] for returning the first and second derivatives of a cubic spline. It also provides a routine for constructing a curve through a given set of data points (where the second derivative at the endpoints is assumed to be zero). The listing within the test program TESTMARKER.PAS at Appendix V provides an example of the use of this unit.

6.9.1 Data types and constants

CONST	:	TNArraySize = 201; (limits array size).
TYPE	:	TNVector = ARRAY[0..TNArraySize] of Float;

6.9.2 Available procedures

Derivative	:	Returns the first derivative of a spline.
SecondDerivative	:	Returns the second derivative of a spline.
CubicSplineFree	:	Constructs a smooth curve through a given set of data points.

6.9.3 Procedure headers

PROCEDURE **Derivative** (NumPoints : INTEGER; VAR XData; NumInter : INTEGER; VAR XInter, YInter);

NumPoints	:	Number of data points.
XData	:	Summed Cubic Spline Free.
NumInter	:	Number of points where the derivative is required.
XInter	:	x values where the derivative is required.
YInter	:	Derivative at x_i (i = 1..NumIter).

PROCEDURE **SecondDerivative** (NumPoints : INTEGER; VAR XData; NumInter : INTEGER; VAR XInter, YInter);

NumPoints	:	Number of data points.
XData	:	Summed Cubic Spline Free.
NumInter	:	Number of points where the derivative is required.
XInter	:	x values where the derivative is required.
YInter	:	Second derivative at x_i (i = 1..NumIter).

PROCEDURE **CubicSplineFree** (NumPoints : INTEGER; VAR XData, YData; NumInter : INTEGER; VAR XInter, YInter; VAR Error : BYTE);

NumPoints	:	Number of data points.
XData	:	Input x data values.
YData	:	Input y data values.
NumInter	:	Number of interpolations.
XInter	:	x coordinates of points at which to interpolate.
YInter	:	Interpolated values at XInter.
Error	:	0 — No Error.
		1 — x values of data points not unique.
		2 — x values of the data points not in ascending order.
		3 — NumPoints < 2.

6.10 Unit FloatDef

The purpose of this unit is to allow the user to specify the type *Float*, either as SINGLE, DOUBLE or EXTENDED. The default value is SINGLE. The unit must be included with all Turbo Pascal code that makes use of the TP/GEM software interface and is edited as required by the user.

6.11 Unit TypeDef

This unit should be included with all source programs that incorporate any code from the TP/GEM interface. There are no functions or procedures, only data types and variables.

6.11.1 Data types

PlotArray	=	ARRAY[1..MaxPt] OF Float;
OutputDisplayType	=	(Screen, GemFile);
OutputDevicesList	=	Set of OutputDisplayType;
GemColorMap	=	ARRAY[0..15] OF INTEGER;
GraphType	=	(Linear, Log, LogLinear, LinearLog);

CONST GemColors : GemColorMap = (0,12,11,14,10,15,13,8,9,4,3,5,2,7,6,1);

6.11.2 **Global variables**

DeviceList	:	OutputDevicesList;
ErrorStr	:	STRING;
ScreenDevice	:	INTEGER;
		— Screen device space covered by the specified plot rectangle
FullScreenDevice	:	INTEGER;
		— Device space covered by full screen
GEMDevice	:	INTEGER;
		— GEM device space covered by the specified plot rectangle
GEMPage	:	INTEGER;
		— Device space covered by full GEM plot area

The user has the option of writing data to the PC screen but not to the GEM output file. The latter option is often more conveniently accomplished during manipulation within GEM Draw or GEM Artline. The *ScreenDevice* variable is useful as an aid for positioning the required data on the screen. For example:

```
MoveTo(DevX(ScreenDevice,0.0)+10,DevY(ScreenDevice,1.0)+20);
OutText('Filter Cutoff : ');
Str(FilterOff:3:1,WriteStr);
OutText(WriteStr);
OutText(' microns');
```

will write the required information near to the top left hand corner of the specified plot rectangle. A number of examples of the application of *ScreenDevice* appear in Appendix II.

6.12 Unit Defaults

The purpose of this unit is to allow the user to tailor the TP/GEM interface to suit his/her own personal preferences. Any value of a global variable written into the procedure *SetDefaults* will overwrite the default value. Note that the unit in which the variable is defined must be called under the *USES* heading within DEFAULTS.PAS which, in turn, must be called within the user source program. Note further that the values of the global variables may be changed at any point within the user source program, as illustrated for example within Appendices I to III.

PROCEDURE **SetDefaults**;

REFERENCES

1. W. Bohm, G. Farin and J. Kahmann, "A Survey of Curve and Surface Methods in CAGD". Computer Aided Geometric Design **1**, 1 (1984).

2. W.H. Press, B.P. Flannery, S.A. Teukolsky and W.T. Vetterling, *Numerical Recipes, the Art of Scientific Computing*, Cambridge University Press (1988).

3. *Turbo Pascal Toolbox Numerical Methods*, Version 4.0, Borland International, Scotts Valley CA (1987).

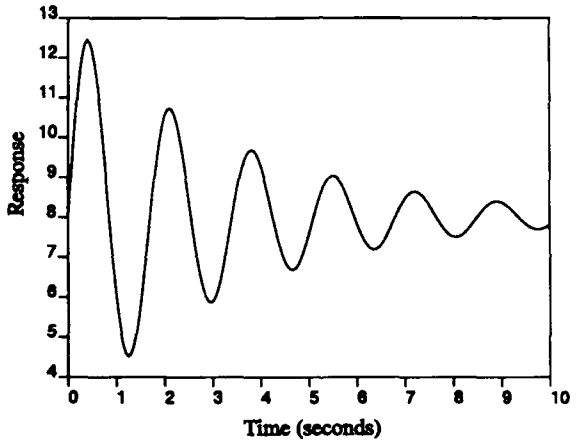
Appendix I

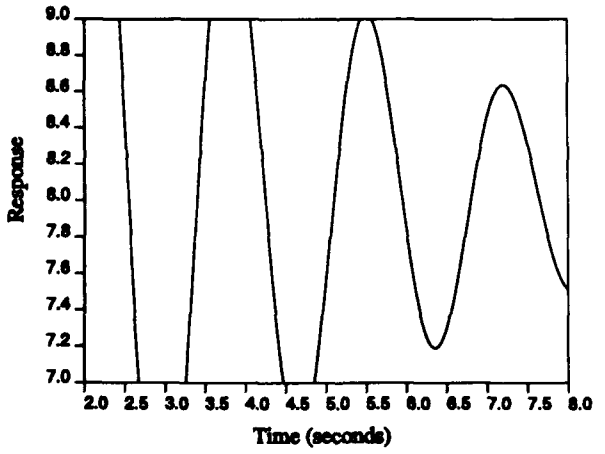
Test Program: EASYPLT.PAS

This is a simple program designed to illustrate the basic use of the TP/GEM interface, with data imported from an external data file. Note the example of the change of the global variable *AxisColor*. This user preference could optionally have been stored in the user defaults unit DEFAULTS.PAS. Automatically scaled and user-selected clipped examples of graphical output are included beneath the code.

```
PROGRAM EASYPLT;
USES
  GRAPH, CRT,
   Coords, Axes, Mathlib, Plotlib, Form, FloatDef, TypeDef, Defaults;
TYPE
  readfile = TEXT;
  ARRAY500 = ARRAY[1..500] of Float;
VAR
  datafile                              : readfile;
  Time, Response                        : array500;
  i, imin, imax, Npts                   : INTEGER;
  TSteps, RSteps, Tdecimals, Rdecimals : INTEGER;
  Tmin, Tmax, Rmin, Rmax:  Float;
  TWorld:  INTEGER;
  ch : CHAR;
BEGIN  {Program}
  ASSIGN(datafile,'PLOT.DAT');
  RESET(datafile);
  i := 0;
  WHILE NOT EOF(datafile) DO
    BEGIN
      i := i + 1;
      READLN(datafile, Time[i], Response[i]);
    END;
  CLOSE(datafile);
  Npts := i;
{
  Plotting Routines Follow
}
  FindExtrema(Time, Npts, Tmin, Tmax, imin, imax);
  FindExtrema(Response, Npts, Rmin, Rmax, imin, imax);
  AxisScale('TIME', TSteps, Tmin, Tmax, TDecimals);
  AxisScale('RESPONSE', RSteps, Rmin, Rmax, RDecimals);
  DefineWorld(TWorld,tmin,Rmin,tmax,Rmax);
{
  Set user-defined defaults and the size of the plot rectangle
}
  SetDefaults;
  AxisColor := LightBlue;
  GEMStart('EASYPLT.GEM',Landscape,0.68);
  ScreenStart(0.65);
```

```
{
  Draw, label and name axes
}
  LabelXaxis(lightgreen,TSteps, Tdecimals, TWorld);
  LabelYaxis(lightgreen,RSteps,Rdecimals, TWorld);
  GemTextSettings(Charter,20);
  NameXaxis(lightgreen, 'Time (seconds)');
  NameYaxis(lightgreen, 'Response');
{
  Plot curve data
}
  PlotCurve(Yellow, Npts, Linear, Time, Response, TWorld);
  ch := ReadKey;
  PlotEnd;
END.
```





Appendix II

Test Program: MULTIPLT.PAS

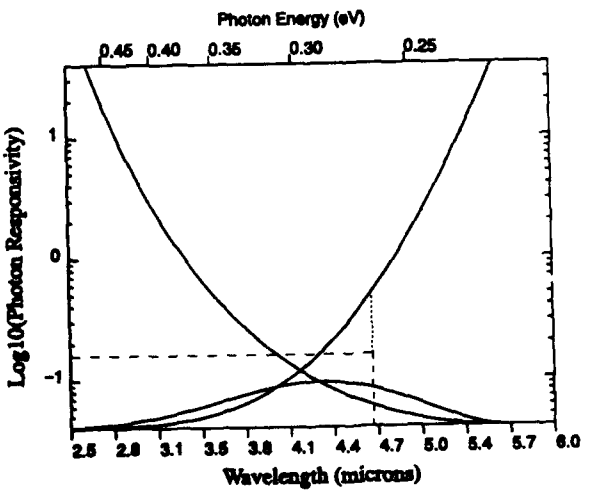
This second example is designed to illustrate a number of features of the TP/Gem interface. The code is a significantly modified version of a genuine program and as such, the actual numbers and plotted curves should not be taken too seriously. Examples of multiple World use, global variable changes, user-defined labelling, NDC dashed lines connecting curves from different Worlds and logarithmic axes are all included together with procedures for writing screen data within the plot rectangle and an application of the unit FORM procedures.

```
PROGRAM MultiPlt;
USES
  GRAPH,CRT,
  Coords, Axes, Mathlib, PlotLib, SBezier, FloatDef, Form, TypeDef;
TYPE
  Array300 = ARRAY[1..300] OF Float;
VAR
  ch                                                        : CHAR;
  ymin, ymax, x1, y1, x2, y2, lmin, lmax, Mmin, Mmax, Tb,
  RqMin, RqMax, lambda1, lambda2, fq, fqRef, RqNorm,
  RqMmin, RqMmax, dlambda, fqFixed, lambdaRef, lambdaFixed,
  RqRef, RqFixed, EffectiveCutoff, CutoffRef,
  CutoffFixed, C1, MCutoffFixed, lambdaCutoff,
  WorkFunction                                              : Float;
  i, j, Npts, lSteps, MSteps, imin, imax, RqMSteps, RqSteps,
  LogWorld, RqWorld, RqMWorld, MWorld, lDecimals, MDecimals,
  RqDecimals, RqMDecimals                                   : INTEGER;
  lambda, lambdaT, FracMlambda, Rq, RqM, wx, wy             : Array300;
  MyLabels, TickPosn                                        : Array100;
PROCEDURE LotsOfCalculations;
{
 The code in this procedure involves calculation of array data that
  is not relevant to illustration of the operation of the TP/GEM
  interface.  The code is not therefore replicated within this
  Appendix.
}
PROCEDURE DisplayStuffOnScreen;
VAR
  TbStr, LowerStr, UpperStr, deltaStr, EnergyStr, C1Str,
  Rqstr, CutoffStr, fqStr, lambdaStr                    : STRING;
BEGIN
{
  Title plot and write data
}
  SetColor(Lightcyan);
  SetTextJustify(LeftText, BottomText);
  SetTextStyle(2, HorizDir, 4);
  MoveTo(DevX(ScreenDevice, 0.0) + 10, DevY(ScreenDevice, 1.0) + 10);
  OutText('Background T : ');
  Str(Tb:6:1, TbStr);
  OutText(TbStr);
  OutText('K');
```

```
  MoveTo(DevX(ScreenDevice, 0.0) + 10, DevY(ScreenDevice, 1.0)+30);
  OutText('Reference Wavelength (1) : ');
  Str(lambdaRef:5:3, lambdaStr);
  OutText(lambdaStr);
  OutText(' microns');
  MoveTo(DevX(ScreenDevice, 0.0) + 10, DevY(ScreenDevice, 1.0) + 40);
  OutText('Reference Responsivity : ');
  Str(RqRef:8:5, Rqstr);
  OutText(Rqstr);
  MoveTo(DevX(ScreenDevice, 0.0) + 10, DevY(ScreenDevice, 1.0) + 50);
  OutText('Effective Cutoff Wavelength : ');
  Str(CutoffRef:5:3, CutoffStr);
  OutText(CutoffStr);
  OutText(' microns');
  MoveTo(DevX(ScreenDevice, 0.0) + 10, DevY(ScreenDevice, 1.0) + 60);
  OutText('fq : ');
  Str(fqRef:8:6, fqStr);
  OutText(fqStr);
  MoveTo(DevX(ScreenDevice, 0.0) + 10, DevY(ScreenDevice, 1.0) + 80);
  OutText('Reference Wavelength (2) : ');
  Str(lambdaFixed:5:3, lambdaStr);
  OutText(lambdaStr);
  OutText(' microns');
  MoveTo(DevX(ScreenDevice, 0.0) + 10, DevY(ScreenDevice, 1.0) + 90);
  OutText('Reference Responsivity : ');
  Str(RqFixed:8:5, Rqstr);
  OutText(Rqstr);
  MoveTo(DevX(ScreenDevice, 0.0) + 10, DevY(ScreenDevice, 1.0) + 100);
  OutText('Effective Cutoff Wavelength : ');
  Str(CutoffFixed:5:3, CutoffStr);
  OutText(CutoffStr);
  OutText(' microns');
  MoveTo(DevX(ScreenDevice, 0.0) + 10, DevY(ScreenDevice, 1.0) + 110);
  OutText('fq : ');
  Str(fqFixed:8:6, fqStr);
  OutText(fqStr);
  MoveTo(DevX(ScreenDevice, 0.0)+ 350, DevY(ScreenDevice, 1.0) + 10);
  OutText('Work Function : ');
  Str(WorkFunction:5:3, EnergyStr);
  OutText(EnergyStr);
  OutText(' eV');
  MoveTo(DevX(ScreenDevice, 0.0) + 350, DevY(ScreenDevice, 1.0) + 20);
  OutText('Cutoff Wavelength : ');
  Str(lambdaCutoff:5:3, CutoffStr);
  OutText(CutoffStr);
  OutText(' microns');
  MoveTo(DevX(ScreenDevice, 0.0) + 350, DevY(ScreenDevice, 1.0) + 30);
  OutText('C1 : ');
  Str(C1:5:3, C1Str);
  OutText(C1Str);
END;
```

```
PROCEDURE SetValues;
BEGIN
  Tb := 290;
  WorkFunction := 0.219;
  C1 := 0.267;
  lambda1 := 1.0;
  lambda2 := 6.5;
  dlambda  := 0.02;
END;
BEGIN
  SetValues;
  OpenForm('PtSi Cutoff evaluation');
  FloatItem(Tb, 1,           'Background temperature (K) :');
  FloatItem(WorkFunction, 3, '        PtSi Work Function :');
  FloatItem(C1, 2,            'PtSi responsivity constant :');
  FloatItem(lambda1, 1,       '      Lower Spectral Bound :');
  FloatItem(lambda2, 1,       '      Upper Spectral Bound :');
  FloatItem(dlambda, 2,        '         Spectral Interval :');
  CloseForm;
  FormControl := 1;
  WHILE (FormControl <> 100) and (FormControl <> 101) DO
    BEGIN
      CASE FormControl OF
      1 : GetRealValue(Tb);
      2 : GetRealValue(WorkFunction);
      3 : GetRealValue(C1);
      4 : GetRealValue(lambda1);
      5 : GetRealValue(lambda2);
      6 : GetRealValue(dlambda);
      END; {CASE}
    END;   {WHILE}
  LotsOfCalculations;
  FindExtrema(lambda, Npts, lmin, lmax, imin, imax);
  FindExtrema(FracMlambda, Npts, Mmin, Mmax, imin, imax);
  FindExtrema(Rq, Npts, RqMin, RqMax, imin, imax);
  FindExtrema(RqM, Npts, RqMmin, RqMmax, imin, imax);
  AxisScale('WAVELENGTH', lSteps, lmin, lmax, lDecimals);
  AxisScale('FRACTIONAL EXITANCE', MSteps, Mmin, Mmax, MDecimals);
  AxisScale('PHOTON RESPONSIVITY', RqSteps, RqMin, RqMax, RqDecimals);
  ymax := RqMax;
  LogScale('LOG TEST',ymin, ymax, 3);  {Only included as an example of use of}
                                       {LogScale, which would normally be}
                                       {applied in association with a log}
                                       {qualifier within Procedure PlotCurve}
  lmin := 2.5;
  lmax := 6.0;                         {These values are user-inspired,}
  Mmax := 0.25;                        {overwriting the auto-scaled values}
  RqMax := 5.0;                        {calculated within AxisScale}
  RqMSteps := MSteps;
  RqMmax := Mmax;
  DefineWorld(RQWorld, lmin, RqMin, lmax, RqMax);
  DefineWorld(MWorld, lmin, Mmin, lmax, Mmax);
  DefineWorld(RqMWorld, lmin, RqMmin, lmax, RqMmax);
  DefineWorld(LogWorld, lmin, ymin, lmax, ymax);
```

```
{
  Set the size of the plot rectangle
}
  GEMStart('MULTIPLT.GEM',Landscape,0.68);
  ScreenStart(0.85);
  LabelXaxis (magenta, lSteps, lDecimals, RqWorld);
  Side := InSide;
  AxisTicks := TwoSidesTicked;
  LogYAxis (lightgreen, LogWorld);
  Side := OutSide;
  AxisTicks := OneSideTicked;
  GemTextSettings(Dutch, 20);
  NameXaxis (lightmagenta, 'Wavelength (microns)');
  NameYaxis (lightgreen, 'Log10(Photon Responsivity)');
  FOR i:= 1 TO 5 DO
  BEGIN
    MyLabels[i] := 0.20 + i*0.05;
    TickPosn[i] := 1.240/MyLabels[i];
  END;
  GemTextSettings(Swiss,17);
  VarLabelTop (lightgreen, 5, TickPosn, MyLabels, 2, lmin, lmax);
  NameTopAxis(lightmagenta, 'Photon Energy (eV)');
  PlotCurve (blue, Npts, Linear, lambda, FracMlambda, MWorld);
  SmoothBezier(Yellow, Npts, 0, Linear, lambda, FracMlambda, MWorld);
  SmoothBezier(lightgreen, Npts, 0, Linear, lambda, Rq, RqWorld);
  SmoothBezier(lightred, Npts, 0, Linear, lambda, RqM, RqMWorld);
{
  Plot dashed lines
}
  NDC(RqWorld, CutoffFixed , 0.0, x1, y1);
  NDC(RqWorld, CutoffFixed, RqNorm, x2, y2);
  PlotDashedLine(Yellow, Dashed, Halfwidth, x1, y1, x2, y2);
  NDC(RqWorld, lmin, RqNorm, x1, y1);
  PlotDashedLine(Yellow, Dashed, Halfwidth, x1, y1, x2, y2);
  NDC(MWorld, CutoffFixed, MCutoffFixed, x1, y1);
  PlotDashedLine(Yellow, ShortDashed, Halfwidth, x1, y1, x2, y2);
  DisplayStuffOnScreen;
  ch := ReadKey;
  PlotEnd;
END.
```



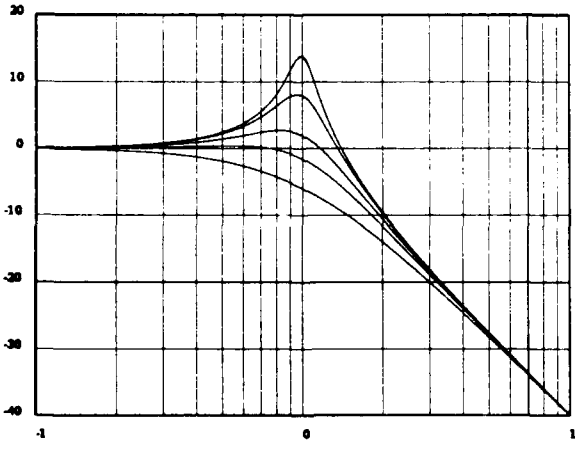
Appendix III

Test Program: GRAFTEST.PAS

GRAFTEST is designed to demonstrate the application of the procedure *SmoothBezier*. For curves that exhibit high curvatures and a large range of curvatures, *SmoothBezier* will provide a significantly better fit than *PlotBezier* at the expense of slower operation and a greater memory requirement. Note that the choice of inside ticks with unit length will produce a complete grid.

```
{
  Program to demonstrate plotting routines. Plots frequency response
  curves of the form:
  20LOG 1.0 / (1.0 - (W/Wn)^2 + 2*ZETA*(W/Wn)) v LOG(W/Wn)
  for W/Wn = 0.1..10.0
  Separate curves for ZETA = 0.1,0.2,0.4,0.6,1.0
}
PROGRAM GrafTest;
USES Graph, Coords, PlotLib, SBezier, Axes, MathLib, FloatDef, TypeDef;
TYPE
  ZTabType = ARRAY[1..5] OF Float;
CONST
  ZTab : ZTabType = (0.1,0.2,0.4,0.6,1.0);
  NPoints = 80;
VAR
  i,j,PlotWorld : INTEGER;
  Zeta,uu,dx,XMin,XMax : Float;
  X,Y,WorkX,WorkY : ARRAY[0..100] OF Float;
  U : ARRAY[1..2] OF Float;  {Complex Number}
BEGIN
  XMin := 0.1;
  XMax := 10.0;
  DefineWorld(PlotWorld,XMin,-40,XMax,20);
  {
    Creates a new world referenced by "PlotWorld"
    World is bounded by (Xmin, -40) on the lower left corner and
    by (XMax, 20) on the upper right.
  }
  GemStart('GRAFTEST.GEM',Landscape,0.75);
  {
    Enables output to GEM meta-file GRAFTEST.GEM. Orientation is landscape
    and 0.75 of the plot area is used for plotting to.
  }
  GemWidth(5);  {Draws GEM lines 5 points wide}
  ScreenStart(0.75); {Plots to 0.75 of the available screen}
  TickLen := 1.0;  {length of ticks on the axis}
  Side := Inside;  {plot ticks on inside of plot area (normally outside)}
  GemTextSettings(Charter, 12); {Text output for gem is Charter font, 12 Point}
  LogXAxis(LightMagenta,PlotWorld); {Draws X Axis in light magenta colour within
                                     World defined by PlotWorld}
  LabelYAxis(LightMagenta,6,0,PlotWorld); {Draws Y Axis in light magenta
                                           in world defined by PlotWorld.
                                           6 Ticks and numbers, No decimals}
```

```
  dx := (Log10(XMax) - Log10(XMin)) / NPoints;
  FOR J := 1 TO 5 DO
    BEGIN
      Zeta := ZTab[J];
      FOR i := 0 TO NPoints DO
        BEGIN
          X[i] := Power(10,i * dx + Log10(XMin));
          U[1] := (1.0 - X[i] * X[i]);
          U[2] := 2.0 * Zeta * X[i];
          uu  := U[1] * U[1] + U[2] * U[2];
          U[1] := U[1] / uu;
          U[2] := U[2] / uu;
          Y[i] := 20 * Log10(SQRT(U[1]*U[1] + U[2] * U[2]));
        END;
      SmoothBezier(Cyan,NPoints+1, j, LogLinear,x,y,PlotWorld);
      {
        Draws a smoothed bezier curve in PlotWorld
        j sets the line type
        Log X axis, Linear Y.
        x, y contain data points
      }
      PlotCurve(LightRed,NPoints,LogLinear,x,y,PlotWorld);
      {
        Draws the curve through x and y.
        Colour is light red,
        NPoints will be plotted.
        Log X axis, Linear Y axis.
      }
    END;
  READLN;
  PlotEnd; {Ends screen graphics and closes GEM meta-file}
  WRITELN('Nonconvergance : ',NonConv);
  WRITELN(ErrorStr);
END.
```



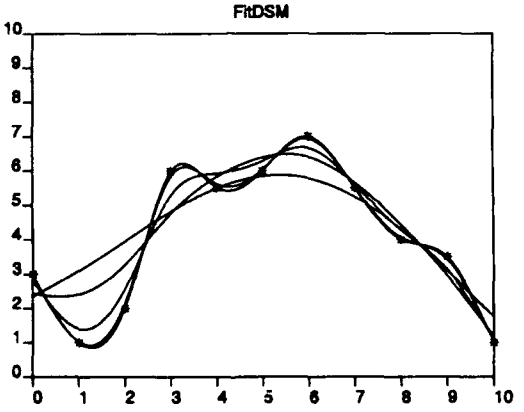
Appendix IV

Test Program: TESTDSM.PAS

TESTDSM provides examples of the use of the discrete smoothing procedures with different values chosen for the smoothing parameter. The procedure *PlotMarkers* is used to mark the individual points defining the original data set.

```
PROGRAM DSMTest;
USES Graph,FloatDef, TypeDef, DSM,Coords,PlotLib,Axes,Bezier, Crt;
VAR
  x,y,b,c,d,ys,wgs : ARRAY[0..10] OF Float;
  u,v,dv,wx,wy : ARRAY[0..100] OF Float;
  i,PlotWorld : INTEGER;
  h,Rho : Float;
  Ch : char;
BEGIN
  {Assign x y and Wgs arrays initial values}
  x[0] := 0.0; y[0] := 3.0; Wgs[0] := 2.0;
  x[1] := 1.0; y[1] := 1.0; Wgs[1] := 1.0;
  x[2] := 2.0; y[2] := 2.0; Wgs[2] := 1.0;
  x[3] := 3.0; y[3] := 6.0; Wgs[3] := 1.5;
  x[4] := 4.0; y[4] := 5.5; Wgs[4] := 1.0;
  x[5] := 5.0; y[5] := 6.0; Wgs[5] := 1.0;
  x[6] := 6.0; y[6] := 7.0; Wgs[6] := 1.5;
  x[7] := 7.0; y[7] := 5.5; Wgs[7] := 1.5;
  x[8] := 8.0; y[8] := 4.0; Wgs[8] := 1.0;
  x[9] := 9.0; y[9] := 3.5; Wgs[9] := 1.0;
  x[10] := 10.0; y[10] := 1.0; Wgs[10] := 2.0;
  FOR i := 0 TO 100 DO
    u[i] := i * 0.1;    {Initialise u array}
  h := 0.1;
  Rho := 0.00001;
  ScreenStart(0.75);  {Start screen graphics}
  GemStart('TESTDSM.GEM',Landscape,0.65);{Start Gem graphics}
  DefineWorld(PlotWorld,0.0,0.0,10.0,10.0); {Define world PlotWorld}
  LabelXAxis(Green,10,0,PlotWorld); {Assign ticks and numbers to X and Y}
  LabelYAxis(Green,10,0,PlotWorld); {axes}
  PlotMarkers(Blue,Star,0.02,11,Linear,x,y,PlotWorld); {Plot stars as markers}
                                                       {at data points for x and y}
  NameTopAxis(Magenta,'FitDSM'); {Labels top axis with "FitDSM"}
  FOR i := 1 TO 5 DO  {five Discrete smoothing curves with different emphasis}
                      {on smoothing (i=1) to curve fitting (i=5)}
    BEGIN
      Rho := Rho * 10;  {Rho determines the emphasis - see manual and above}
      FitDSM(h,Rho,11,x,y,Wgs,ys,b,c,d); {compute discrete cubic spline}
      FitCS(11,101,u,v,x,ys,b,c,d); {Evaluate cubic spline at each point}
      FitDS(11,101,u,dv,x,ys,b,c,d);{Evaluate first derivative of cubic}
                                    {spline at each point (not plotted)}
      PlotCurve(i, 101, Linear, u, v, PlotWorld); {Plot the new curve}
    END;
```

```
  PlotBezier(Yellow,101,0,Linear,u,v,wx,wy,PlotWorld);
  {
    Plot the final curve using beziers.
    Colour is Yellow.
    Plots 101 points, No dashes (NDashes = 0).
    Plot is against a linear axis.
    Curve is plotted through points defined by arrays u and v.
    wx and wy are working arrays. These are used by PlotBezier.
    Curve is plotted in world defined by PlotWorld.
  }
  While not KeyPressed Do;
  Ch := ReadKey; {clear the keyboard buffer}
  PlotEnd;
END.
```

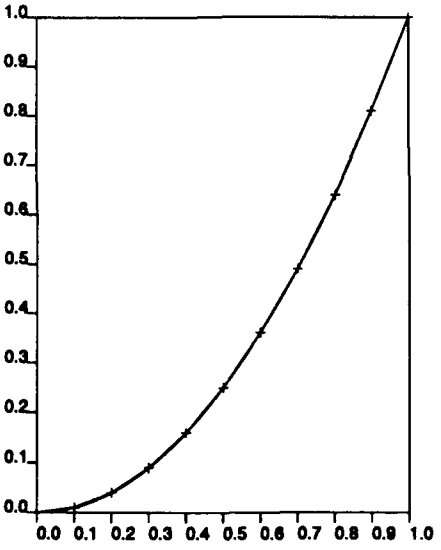


Appendix V

Test Program: TESTMARKER.PAS

```
PROGRAM TestMarker;
USES Graph,PlotLib,FloatDef,TypeDef, Axes,Coords,Interp;
VAR
  x,y : TNvector;
  xx,yy : ARRAY[0..200] OF Float;
  i,PlotWorld : INTEGER;
  Error : BYTE;
BEGIN
  FOR i := 0 TO 10 DO
    BEGIN
      x[i+1] := i * 0.1;
      y[i+1] := x[i+1]*x[i+1];
    END;
  FOR i := 0 TO 100 DO xx[i] := i * 0.01;
  CubicSplineFree(11,x,y,101,xx,yy,Error);
  {
    takes 11 data points from arrays x and y and returns them
    as 101 data points in arrays xx and yy.
  }
  ScreenStart(0.75);
  {
    Initialise screen graphics using 0.75 of the screen area
  }
  GemStart('TESTMARK.GEM',Portrait,0.70);
  {
    Initialise GEM plotting. Orientation Portrait, uses 0.7 of the plot
    area. Gem output will be sent to file TEST.GEM
  }
  DefineWorld(PlotWorld,0.0,0.0,1.0,1.0);
  {
    Define a world indexed by PlotWorld.
    lower left is (0,0), upper right (1, 1)
  }
  LabelXAxis(Magenta,10,1,PlotWorld);
  {
    Draw 10 ticks and numbers in magenta along x axis. colour is magenta.
    Displays 1 decimal point. Drawn in world defined by PlotWorld.
  }
  LabelYAxis(Magenta,10,1,PlotWorld);
  {
    Draw 10 ticks and numbers in magenta along y axis. colour is magenta.
    Displays one decimal point. Drawn in world defined by PlotWorld.
  }
  PlotCurve(Cyan,11,Linear,x[1],y[1],PlotWorld);
  {
    Draws a cyan curve of x vs y against a linear axis. Curve is drawn in
    world defined by PlotWorld.
  }
```

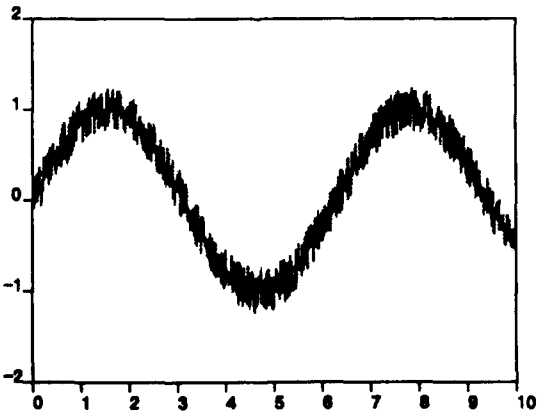
```
  PlotMarkers(LightMagenta,Plus,0.02,11,Linear,x[1],y[1],PlotWorld);
  {
    Draws a "+" as a marker at each of the data points described by arrays
    x and y. Markers are light magenta, drawn in world PlotWorld against
    a linear axis.
    Plots 11 Markers at scale 0.02 of normal size of standard device
  }
  PlotCurve(Green,101,Linear,xx,yy,PlotWorld);
  {
    As with above PlotCurve only plots xx and yy arrays which were returned
    by CubicSplineFree. Careful examination shows the difference between
    the smoothed curve and the plotted original data.
  }
  READLN;
  PlotEnd; {end screen and gem graphics}
END.
```



Appendix VI

Test Program: SMOOTHTEST.PAS

```
{************************************************************************}
{ PROGRAM to test the smoothing routine Smoothft.                        }
{************************************************************************}
PROGRAM SmoothTest;
USES Graph,Smooth,PlotLib, Coords, FloatDef,TypeDef, Axes;
VAR
  x,y,z : ARRAY[1..1024] OF Float;
  dx : Float;
  i,PlotWorld : INTEGER;
BEGIN
  RANDOMIZE;  {Seed random number generator}
  dx := 0.01;
  FOR i := 0 TO 1000 DO
    BEGIN
      x[i+1] := i * dx;
      z[i+1] := SIN(x[i+1]);                  {sine values, x versus z}
      y[i+1] := z[i+1] + 0.5 * (Random-0.5); {sine values from z with random}
                                              {noise added}
    END;
  GEMStart('SMOTEST.GEM',Landscape,0.68);
  ScreenStart(0.75); {Start Screen Graphics}
  DefineWorld(PlotWorld,0,-2,10,2); {Define a world to plot in}
  LabelXAxis(Red,10,0,PlotWorld);   {Put numbers and ticks on X axis}
  LabelYAxis(Red,4,0,PlotWorld);    {Put numbers and ticks on Y axis}
  PlotCurve(Magenta,1001,Linear,x,z,PlotWorld); {Plot sine curve x vs z}
  PlotCurve(Red,1001,Linear,x,y,PlotWorld); {Plot sine curve with "noise"}
  Smoothft(y,1001,100);
   {
     takes array y for smoothing.
     smooths 1001 points,
     smooths over 100 points.
   }
  PlotCurve(Green,1001,Linear,x,y,PlotWorld);  {Plots the smoothed curve}
                                               {returned in y}
  READLN;
  PlotEnd; {End screen graphics}
END.
```



Appendix VII

Test Program: FORMTEST.PAS

```
PROGRAM FormTest;
USES
  Crt, Form, FloatDef;
VAR
  SomeNumber                    : INTEGER;
  Lambda1, Lambda2, DLambda,
  Tb, CutOn, WorkFunction, C1 : Float;
  astring                       : String;
PROCEDURE SetValues;
BEGIN
  Tb            := 290;
  WorkFunction  := 0.213429;
  C1            := -0.267E-09;
  CutOn         := 3.4;
  Lambda1       := 1.0;
  Lambda2       := 6.5;
  DLambda       := 0.02;
  SomeNumber    := 8;
  astring       := 'Hello World';
END;
PROCEDURE DisplayForm;
{
  This Procedure will set up the screen display for a desired input form.
  Procedures used are -
    OpenForm - Will display at the top of the screen the desired heading.
               The heading is passed as a parameter within quotes.
    AdvanceLine - Allows a blank line to be inserted on screen.
    CloseForm - Displays a footer for the input form. It displays the
                message 'Press PgDn OR ESC to Accept Values'.
    FloatItem,
    IntegerItem,  Used for placing prompts on screen, Stating what variables
    StringItem    are associated with each prompt and the number of decimal
                  points to be displayed with each variable
     - Call procedure FloatItem, IntegerItem or StringItem depending
       upon what sort of data item is on a given line.
     - Parameters to the function calls are (in order)
       1. Variable to store the input data.
       2. Number of Decimal Points to display. For Integer and String
          Values this number is ignored.
       3. The prompt to appear for the value at a given line.
          Note that the prompt will commence in Column 5. The cursor will
          be placed one space after the end of the prompt.
    A form should be built up in the following manner
       1 - OpenForm('Form Heading');
       2 -  FloatItem
         or IntegerItem
         or StringItem
```

```
            These should appear on each line where there will be
            an input prompt. They may be interspersed with a call to
            the AdvanceLine procedure in order to improve readability
            or group related items (or both). They should be in the
            order data would normally be entered at the prompts
        3 - CloseForm;
        4 - Having built up the screen it is necessary to add the code
            to read the data values.
            To do this code must be inserted between the beginning and end
            of the Case Statement.
            It takes the following format :
               FormControl := 1;    or whichever field cursor goes in first
               Case FormControl of
               --- Start inserting code here as needed ---
               1 : GetRealValue(Tb);
               2 : GetIntegerValue(C2);
                 .
                 .
               N : GetStringValue(InputString);
               --- after the last prompt no further code is required ---
               End;
            Each prompt must be assigned a number which corresponds to
            the position of the prompt in the form layout defined above.
            First Prompt is 1
            Second is 2
            and so on. The number MUST be followed by the ':' character.
            note that the 'Get' procedures accept parameters which
            correspond to the variables defined above
        5 - The form should now be complete. To run it, call DisplayForm from
            your application.
}
BEGIN
  OpenForm(' PtSi CUTOFF EVALUATION');
  FloatItem(Tb, 1,            '            Background Temperature (K):');
  FloatItem(WorkFunction, 3,  '              PtSi Work Function (eV) :');
  FloatItem(C1, 2,            '        PtSi Responsivity Constant C1 :');
  AdvanceLine;
  FloatItem(CutOn, 1,         '           Cuton Wavelength (microns) :');
  FloatItem(Lambda1, 1,       '       Lower Spectral Bound (microns) :');
  FloatItem(Lambda2, 1,       '       Upper Spectral Bound (microns) :');
  FloatItem(DLambda, 2,       '         Required Step Size (microns) :');
  AdvanceLine;
  IntegerItem(SomeNumber, 2,  '                Some Number (< 10)   :');
  AdvanceLine;
  StringItem(astring, 0,      '                 Enter a String here :');
  CloseForm;
```

```
  FormControl := 1;
  While (FormControl <> 100) and (FormControl <> 101) Do
    Begin
      Case FormControl of
      1 : GetRealValue(tb);
      2 : GetRealValue(WorkFunction);
      3 : GetRealValue(C1);
      4 : GetRealValue(CutOn);
      5 : GetRealValue(Lambda1);
      6 : GetRealValue(Lambda2);
      7 : GetRealValue(DLambda);
      8 : Begin                                {Example of validity checking}
            GetIntegerValue(SomeNumber);
            IF SomeNumber >= 10 THEN      {Value out of range}
              BEGIN
                GoToXY(10,24);
                Write('This number should be less than 10');
                FormControl := 8;
              END
            ELSE                  {Ensure Error message disappears}
              BEGIN
                GoToXY(10,24);
                Write('                                    ');
              END;
          End;
      9 : GetStringValue(astring);
      END; {Case}
    End;
END;
BEGIN
  SetValues;
  DisplayForm;
END.
```

DISTRIBUTION — No. of Copies

DEPARTMENT OF DEFENCE

Defence Science and Technology Organisation

Chief Defence Scientist First Assistant Secretary, Science Policy Director General Science & Technology Programs	1
Counsellor Defence Science London	Control Sheet Only
Counsellor Defence Science Washington	Control Sheet Only
Defence Science Representative, Bangkok	Control Sheet Only
Scientific Adviser, Defence Research Centre, Kuala Lumpur	Control Sheet Only

Aeronautical Research Laboratory

Director, Aeronautical Research Laboratory	1
Chief, Flight Mechanics and Propulsion Division	1
Chief, Aircraft Materials Division	1
Chief, Aircraft Structures Division	1
Chief, Aircraft Systems Division	1

Electronics Research Laboratory

Director, Electronics Research Laboratory	1
Chief, Communications Division	1
Chief, Electronic Warfare Division	1
Chief, Information Technology Division	1

Materials Research Laboratory

Director, Materials Research Laboratory	1
Chief, Explosives Division	1
Chief, Materials Division	1
Chief, Protective Chemistry Division	1
Chief, Underwater Weapon and Countermeasure Systems Division	1
Chief, Scientific Services Division	1
Chief, Engineering Support Division	1

Surveillance Research Laboratory

Director, Surveillance Research Laboratory	1
Chief, High Frequency Radar Division	1
Chief, Microwave Radar Division	1
Chief, Optoelectronics Division	1

Weapons Systems Research Laboratory

Director, Weapons Systems Research Laboratory	1
Chief, Combat Systems Division	1
Dr M. Davies	1
Chief, Guided Weapons Division	1
Chief, Maritime Systems Division	1
Chief, Ordnance Systems Division	1
Research Leader, Guided Weapons	1
Research Leader, Seeker Technology	1
Head, Advanced Seekers	1
Head, Dynamics and Trials	1
Head, Guidance, Control and Simulation	1
Head, Guided Weapon Projects	1

Head, Guided Weapons Engineering	1
Head, Guided Weapons Projects	1
Head, Radio Frequency Seekers	1
Head, System Performance Analysis	1
Mr R.D. Anderson	1
Dr D. Bucco	1
Mr S.C.B. Garner	1
Mr D.J. Hards	1
Mr R.P. Johnson	1
Mr J.P. Owen	1
Dr R.L. Pope	1
Dr M.B. Pszczel	1
Mr J. Repo	1
Mr Scott G. Simmonds	1
Mr G.B. Thamm	1
Mr R.M. Thamm	1
Dr O.M. Williams	1
Scientific Adviser, Defence Intelligence Organisation	1
Director of Departmental Publications, for AGPS	1
Air Office	
Air Force Scientific Adviser	1
Navy Office	
Navy Scientific Adviser	Control Sheet Only
Army Office	
Army Scientific Adviser	1
Libraries and Information Services	
Defence Library, Technical Reports Centre, Campbell Park	1
Library, Materials Research Laboratory	1
Manager, Document Exchange Centre, Defence Information Services Branch	1
National Library of Australia	1
United Kingdom, Defence Research Information Centre	2
Canada, Director Scientific Information Services	1
New Zealand, Ministry of Defence	1
United States, Defense Technical Information Center	2
Library, Defence Science & Technology Organisation Salisbury	2
Australian Defence Force Academy Library	1
Library, DSD	1
British Library, Document Supply Centre	1
Spares	11
Total No. of Copies	80

DOCUMENT CONTROL DATA SHEET

Security classification of this page : UNCLASSIFIED

1 DOCUMENT NUMBERS

AR Number : AR-006-485

Series Number : WSRL-GD-02/91

Other Numbers :

2 SECURITY CLASSIFICATION

a. Complete Document : Unclassified

b. Title in Isolation : Unclassified

c. Summary in Isolation : Unclassified

3 DOWNGRADING / DELIMITING INSTRUCTIONS

4 TITLE

TURBO PASCAL/GEM SOFTWARE INTERFACE FOR SCIENTIFIC GRAPH PREPARATION

5 PERSONAL AUTHOR (S)

R.M. Thamm,
D.A. Green and
O.M. Williams

6 DOCUMENT DATE

April 1991

7 7.1 TOTAL NUMBER OF PAGES 42

7.2 NUMBER OF REFERENCES 3

8 8.1 CORPORATE AUTHOR (S)

Weapons Systems Research Laboratory

8.2 DOCUMENT SERIES and NUMBER

General Document
02/91

9 REFERENCE NUMBERS

a. Task :

b. Sponsoring Agency :

10 COST CODE

11 IMPRINT (Publishing organisation)

Defence Science and Technology Organisation

12 COMPUTER PROGRAM (S) (Title (s) and language (s))

13 RELEASE LIMITATIONS (of the document)

Approved for Public Release.

Security classification of this page : UNCLASSIFIED

Security classification of this page : UNCLASSIFIED

14 ANNOUNCEMENT LIMITATIONS (of the information on these pages)

No limitations

15 DESCRIPTORS		16 COSATI CODES
a. EJC Thesaurus Terms	Graphs Turbo Pascal (programming language)	1205
b. Non - Thesaurus Terms	GEM software Scientific graph preparation	

17 SUMMARY OR ABSTRACT
(if this is security classified, the announcement of this report will be similarly classified)

(U) This document represents a user manual for the Turbo Pascal/GEM software interface that has been developed recently within Guided Weapons Division. The interface has been developed primarily to enable Turbo Pascal programmer to replicate their softcopy graphical output in high quality hardcopy form. The output from the interface is a GEM file that can be edited as required within either GEM Draw or GEM Artline, prior to generation of a PostScript file. A number of mathematical procedures have been included in order to extend the capabilities of Turbo Pascal, together with a useful library of curve fitting procedures. The interface also incorporates a form filling technique for versatile data entry at any point within a Turbo Pascal program. A detailed description of all procedures and functions is included together with the code and graphical output from a number of test programs designed to illustrate the capabilities of the interface.